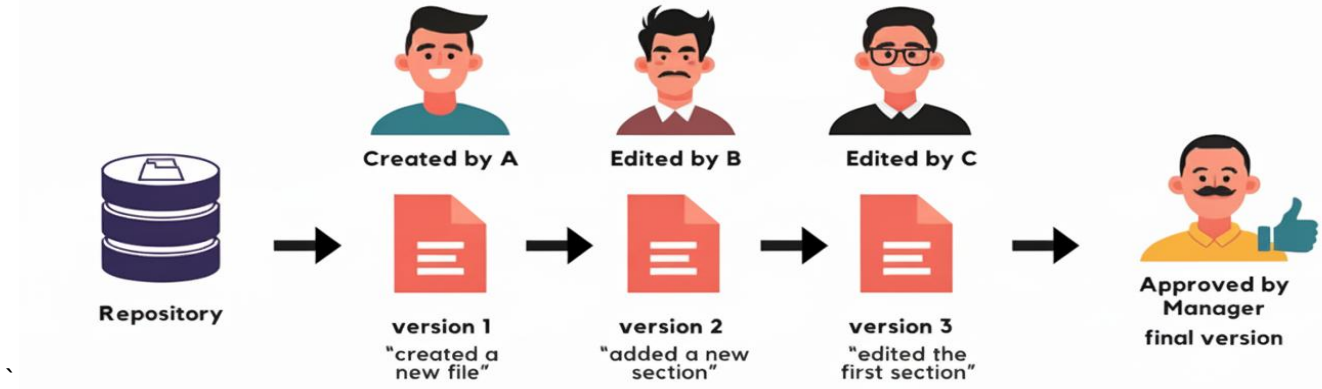
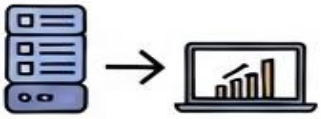
VERSION CONTROL

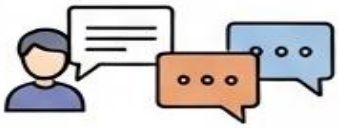


Save project Commits

Branches

Test ideas safely


Centralized vs Git (Distributed)

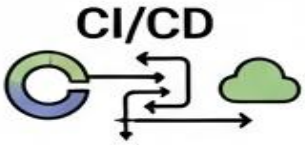

Workflow

Pull Requests & code reviews

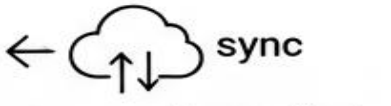
Pull requests Feature Branching

Workflow

Auto test & deploy


CI/CD

CI/CD Platforms



Powers massive projects

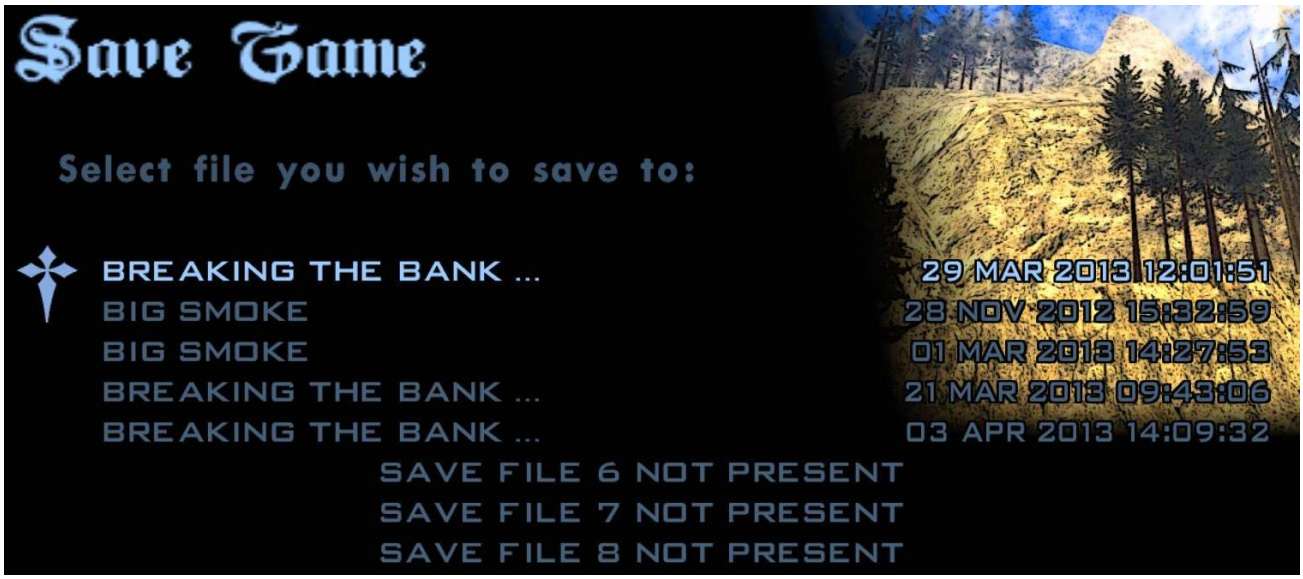
Infrastructure from Git

Azure Repos

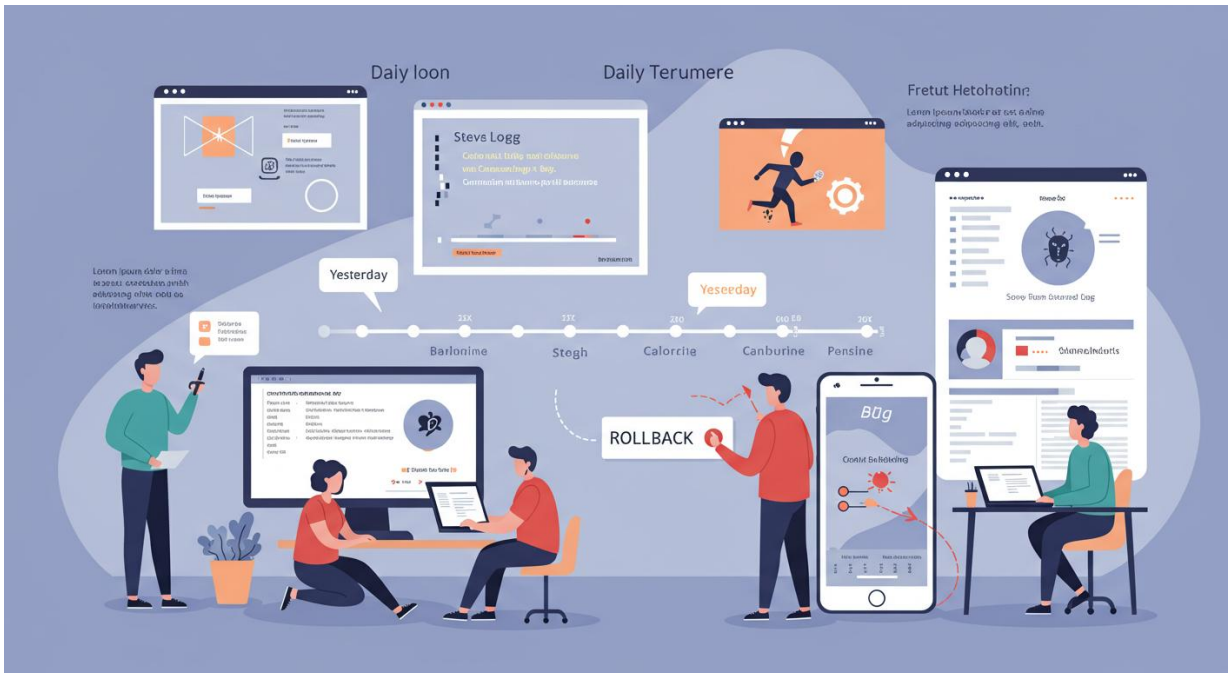
🌱 1. Introduction to Version Control

✅ Technical Definition:
Version control is a system that records changes to files over time, allowing developers to track history, revert to earlier versions, and collaborate without overwriting each other's work.

🎀 Explanation:
It's like saving checkpoints in a video game 🎮 — if your character dies or you mess up, you can reload from the last save instead of starting from scratch (context: commits = save points).



🌍 Real-World Example:
A team building a fitness tracking app saves code changes daily. If a bug appears, they can roll back to yesterday's working version.



🧩 2. Centralized vs Distributed Systems

✅ **Technical Definition:**

- **Centralized VCS (CVCS)**: One central server stores the code, and all team members connect to it (e.g., Subversion, Perforce).
- **Distributed VCS (DVCS)**: Every developer has a full copy of the repository locally (e.g., Git, Mercurial).

🎀 **Explanation:**

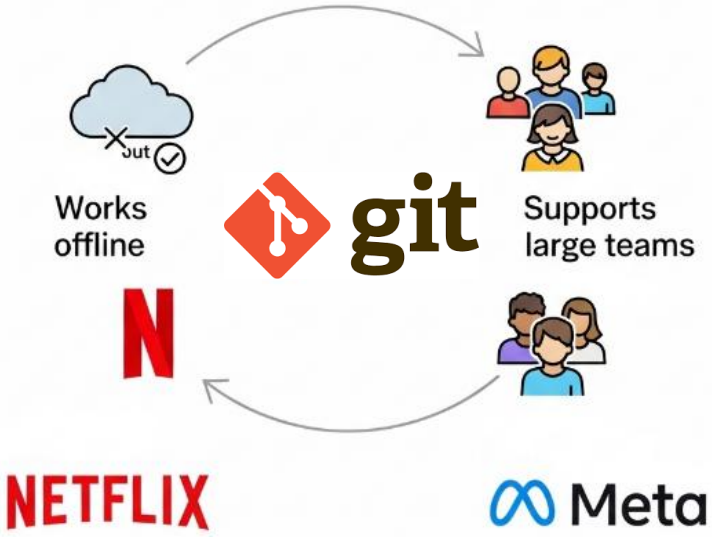
🛒 **Centralized Version Control**

One **shared grocery list** 📝 stuck on the fridge 🧊 at home.
Everyone in the family must go to the fridge to check ✅ or add items.
If the fridge note gets lost ❌ (server down), no one knows what to buy.

📱 **Distributed Version Control**

Each family member has their **own copy of the grocery list** on their phone 📲.
They can add 🍎 🥛 items anytime, even if they're not at home.
Later, they sync 🔄 with the family list so no one forgets the eggs 🥚

🌍 **Real-World Example:**
Most modern companies (Netflix, Facebook) use Git (DVCS) because it works offline and supports large teams.

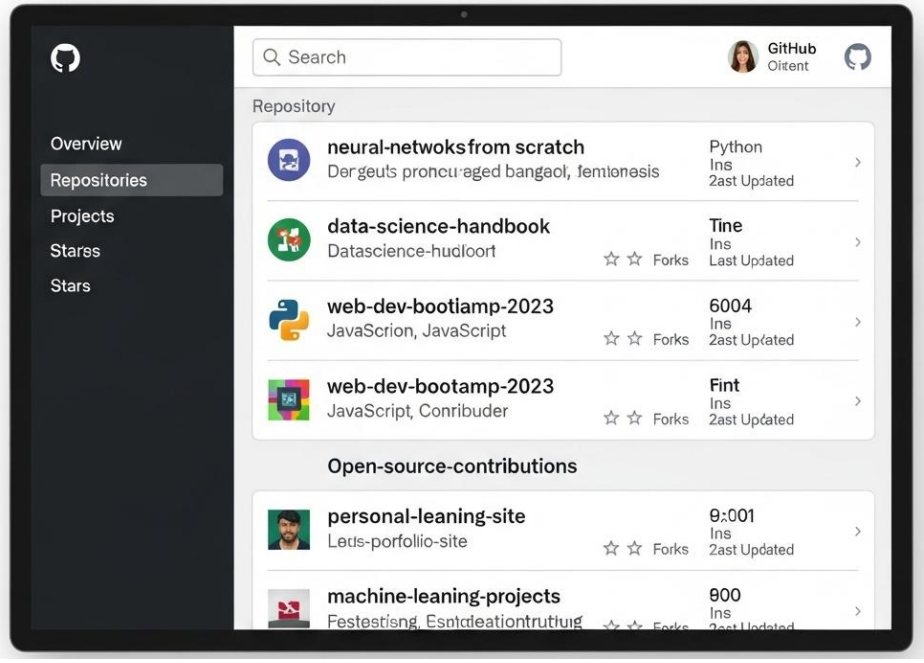


🏗️ 3. Core Concepts

3.1 Repository (Repo)

✅ **Definition:** A storage space for your project's files and history.
🎀 **Explanation:** Photo Album 📸 – A repo is like your photo album where you keep all your pictures. You can go back and see old ones (history), add new ones (commits), or organize them into different albums (branches).

🌍 **Example:** A GitHub repo for your app stores code, images, and documentation.

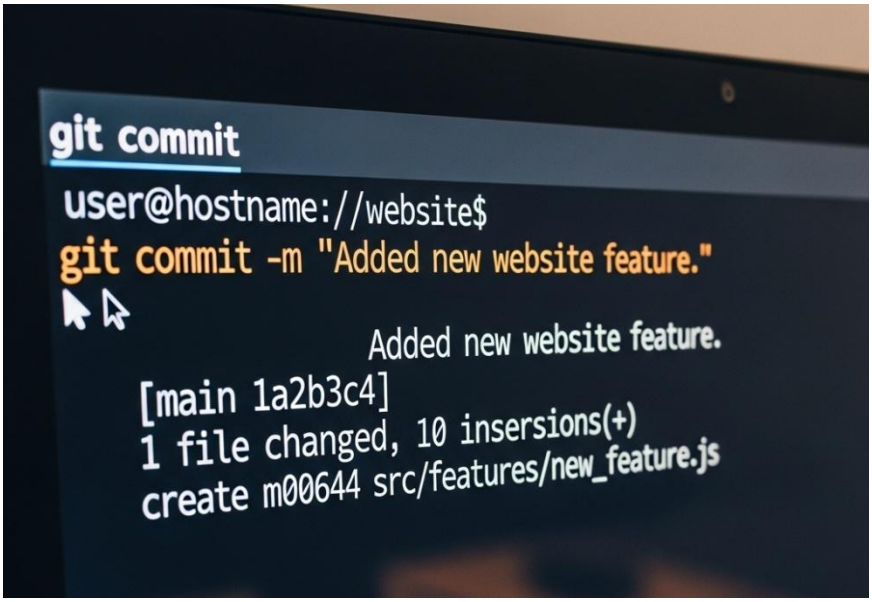


3.2 Commits

✅ **Definition:** Snapshots of changes saved in the repo.

🎀 **Explanation:** Like saving progress in a game 🎮 — you can return anytime without losing your level.

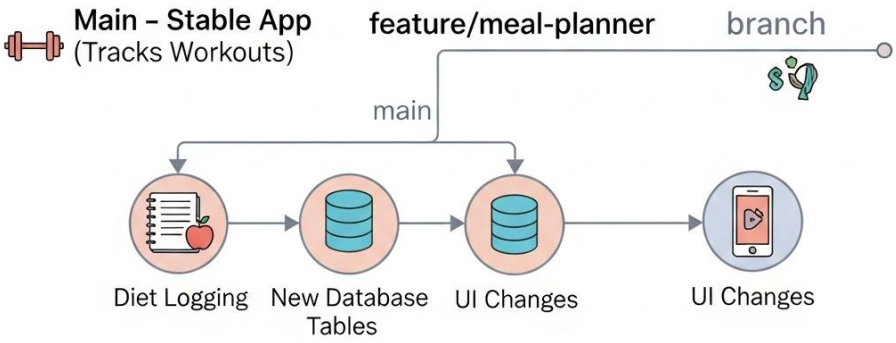
🌍 **Example:** A commit that says “Added new website feature.”



3.3 Branches

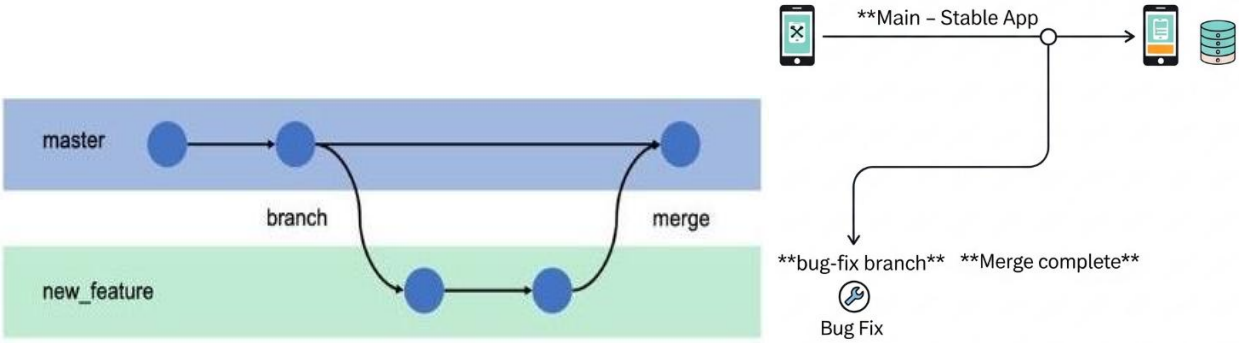
✅ **Definition:** Separate lines of development.
🎀 **Explanation:** Like a writer keeping a storyboard — each thread of ideas runs separately. If one plotline doesn’t work, it won’t ruin the main story. ✍️ 📖

🌎 **Example:** The meal planner feature was built in a separate branch, tested safely, and then merged back into the main fitness app without breaking it.”



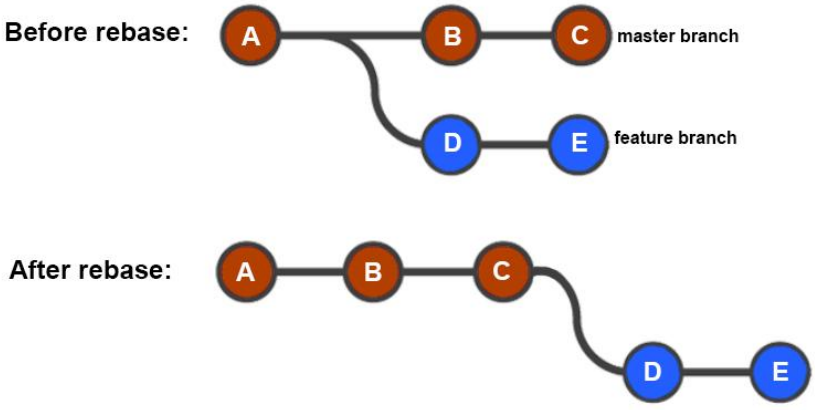
3.4 Merge

✅ **Definition:** Combining changes from one branch into another.
🎀 **Explanation:** Like blending two smoothie recipes 🍓 🍋 into one.
🌎 **Example:** Merging a new feature or bug fix branch into main.



3.5 Rebase

✅ **Definition:** Taking commits from one branch and replaying them on top of another branch.
🎀 **Explanation:** Like rewriting your diary so it reads smoothly without messy scribbles 📓.
🌎 **Example:** Rebasing feature/chat onto the latest main before merging.



🌳 4. Branching Strategies

✅ **Technical Definition:**
Different ways teams structure branches to manage workflows. Popular strategies include:

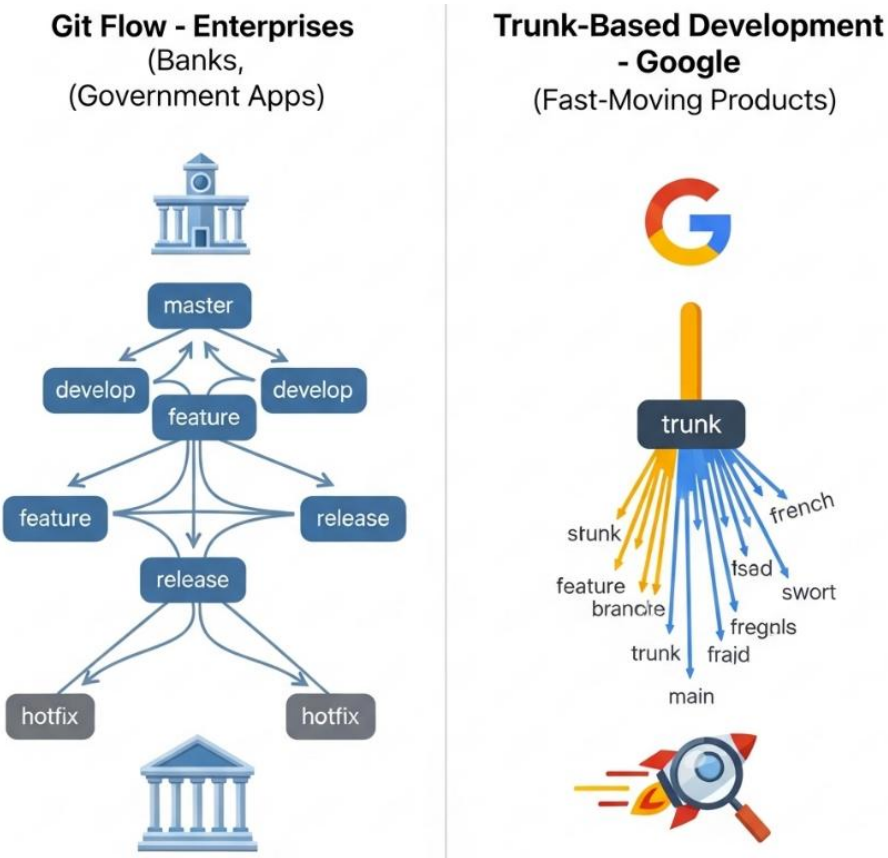
- **Feature Branching**: Separate branch for each feature.
- **Git Flow**: Dedicated develop, main, release, and hotfix branches.
- **Trunk-Based Development**: All developers commit to main frequently.

🎀 **Explanation:**
It's like group projects in school:

- Feature branching = everyone works on their own essay section.
- Git Flow = some write drafts, others polish, then final submission.
- Trunk-based = everyone edits the same Google Doc live (fast but risky).

🌍 **Real-World Example:**

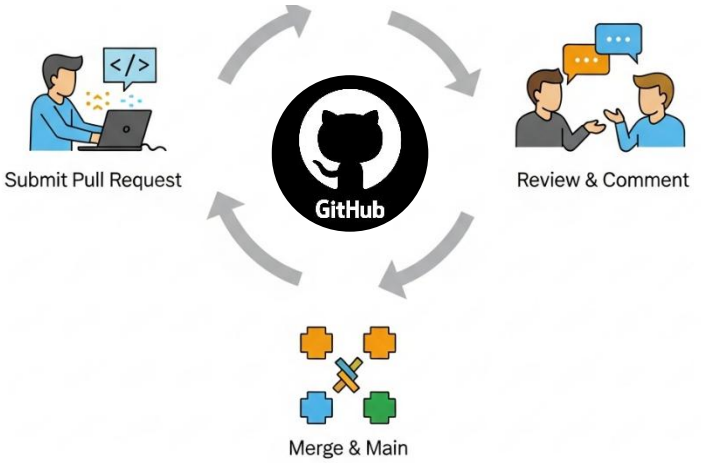
- **Git Flow** is common in enterprises (banks, government apps).
- **Trunk-Based Development** is used at **Google** for fast-moving products.



🤝 5. Collaboration & Workflows

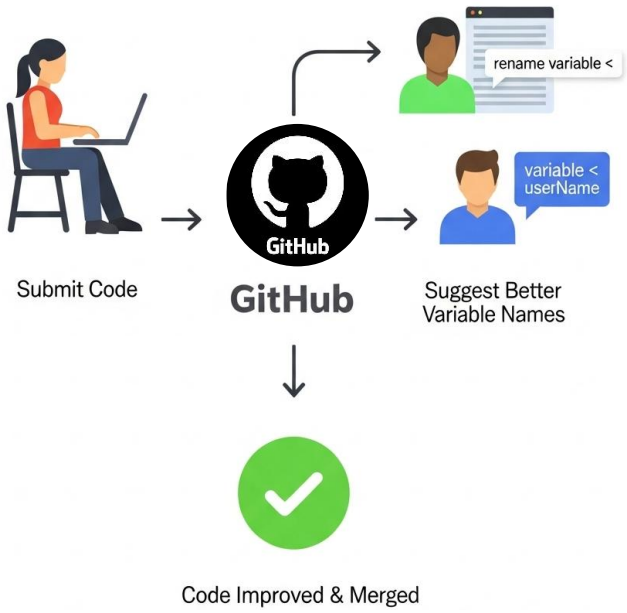
Pull Requests (PRs)

✅ **Definition:** A request to merge code after review.
🎀 **Explanation:** Like asking a friend to proofread your Instagram caption before posting 📸.
🌍 **Example:** On GitHub, teammates review and comment on code changes before merging.



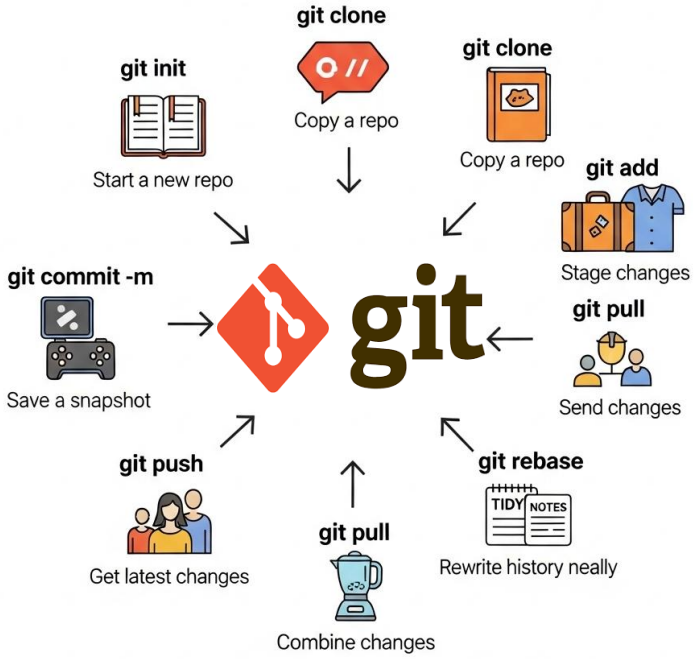
Code Reviews

✅ **Definition:** Peers check each other's code for quality.
🎀 **Explanation:** Like having a gym buddy spot your workout 💪 to make sure you don't hurt yourself.
🌍 **Example:** A reviewer suggests improving variable names for clarity.



☸️ 6. Essential Commands (Git)

- **git init** → Start a new repo (*open a new diary* 📔).
- **git clone** → Copy a repo from GitHub (*get your own copy of the recipe book*).
- **git add** → Stage changes (*decide which clothes go in the suitcase*).
- **git commit** -m "message" → Save a snapshot (*lock in progress like a game save*).
- **git push** → Send changes to remote repo (*share updates with friends*).
- **git pull** → Get the latest changes (*sync your diary with others*).
- **git merge** → Combine changes (*blend ideas into one*).
- **git rebase** → Rewrite history neatly (*tidy your notes*).

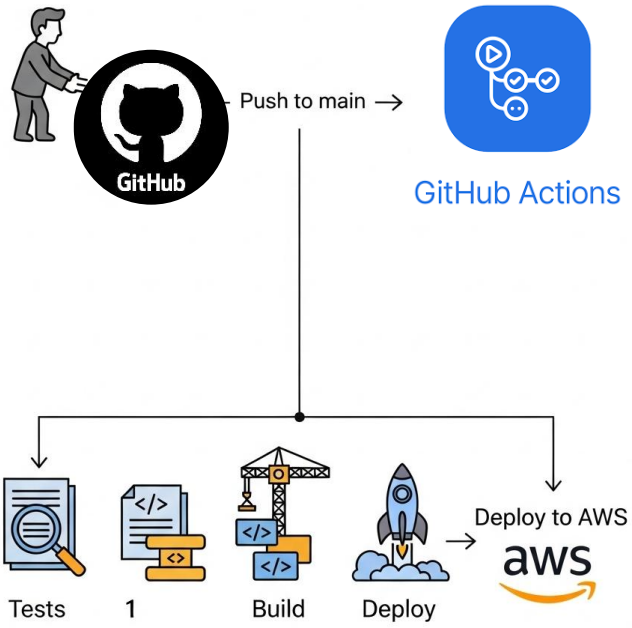


🚀 7. CI/CD Integration with Version Control

✅ **Technical Definition:**
Continuous Integration/Deployment tools hook into Git to automatically build, test, and deploy code when new commits are pushed.

🎀 **Explanation:**
Like having a personal assistant who checks your essay for grammar, prints it, and submits it whenever you make edits ✍️.

🌍 **Real-World Example:**
On GitHub, a push to main triggers GitHub Actions to run tests and deploy to AWS.

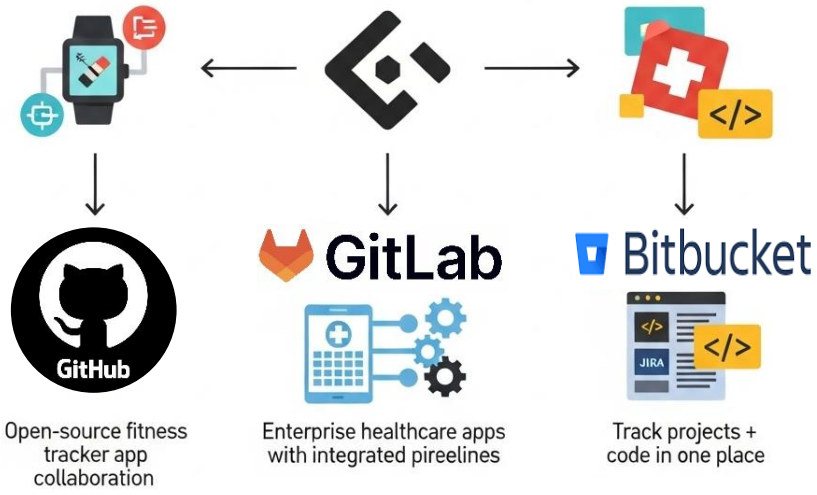


🏢 8. Platforms

- **GitHub**: Most popular, open source + enterprise.
- **GitLab**: Integrated CI/CD pipelines.
- **Bitbucket**: Atlassian's tool, integrated with Jira.
- **Azure Repos**: For Microsoft ecosystems.

🌍 **Example:**

- **GitHub** → Open-source fitness tracker app collaboration.
- **GitLab** → Enterprise teams building healthcare apps with integrated pipelines.
- **Bitbucket** → Jira users tracking project + code in one place.



---

9. Advanced Practices

GitOps

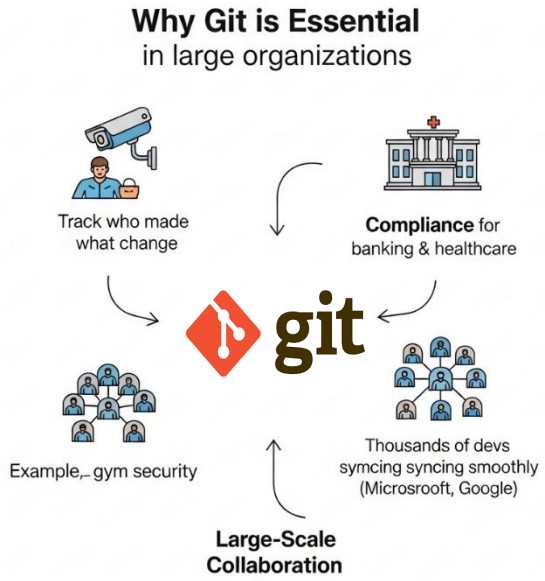
✅ **Definition:**
Using Git as the single source of truth for infrastructure and deployments.

🎀 **Explanation:**
Like keeping all your workout and diet plans in one notebook 📓, and anytime you update it, your smart fridge and smartwatch automatically adjust.

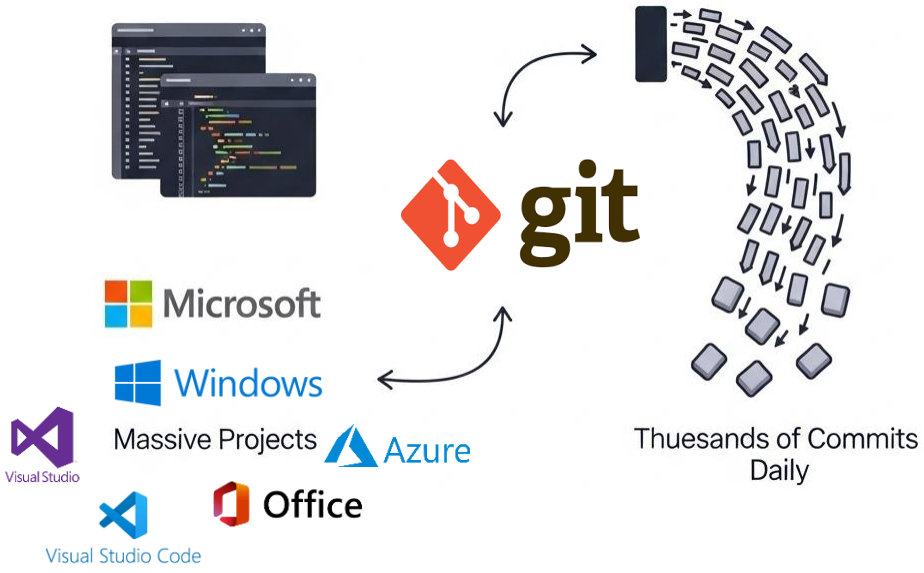
🌍 **Example:**
Companies like **Weaveworks** and **Amazon** use GitOps to manage Kubernetes clusters — changing code in Git automatically updates cloud infrastructure.

🔑 10. Enterprise Use Cases

- **Auditability**: Track who made what change (like keeping CCTV in your gym).
- **Compliance**: Required for banking, healthcare.
- **Large-Scale Collaboration**: Thousands of devs syncing smoothly (e.g., Microsoft, Google).



🌍 **Example:**
Microsoft manages massive projects like Visual Studio using Git, with thousands of commits daily.



🆕 11. Git Hooks

✅ **Technical Definition:**
Custom scripts that run automatically before or after Git events (like commits, pushes, merges) to enforce rules or automate tasks.

🎀 **Down-to-Earth Explanation:**
It's like having a personal assistant check your work before you leave the house 👩‍💼 — "Did you lock the door? Did you take your keys?"

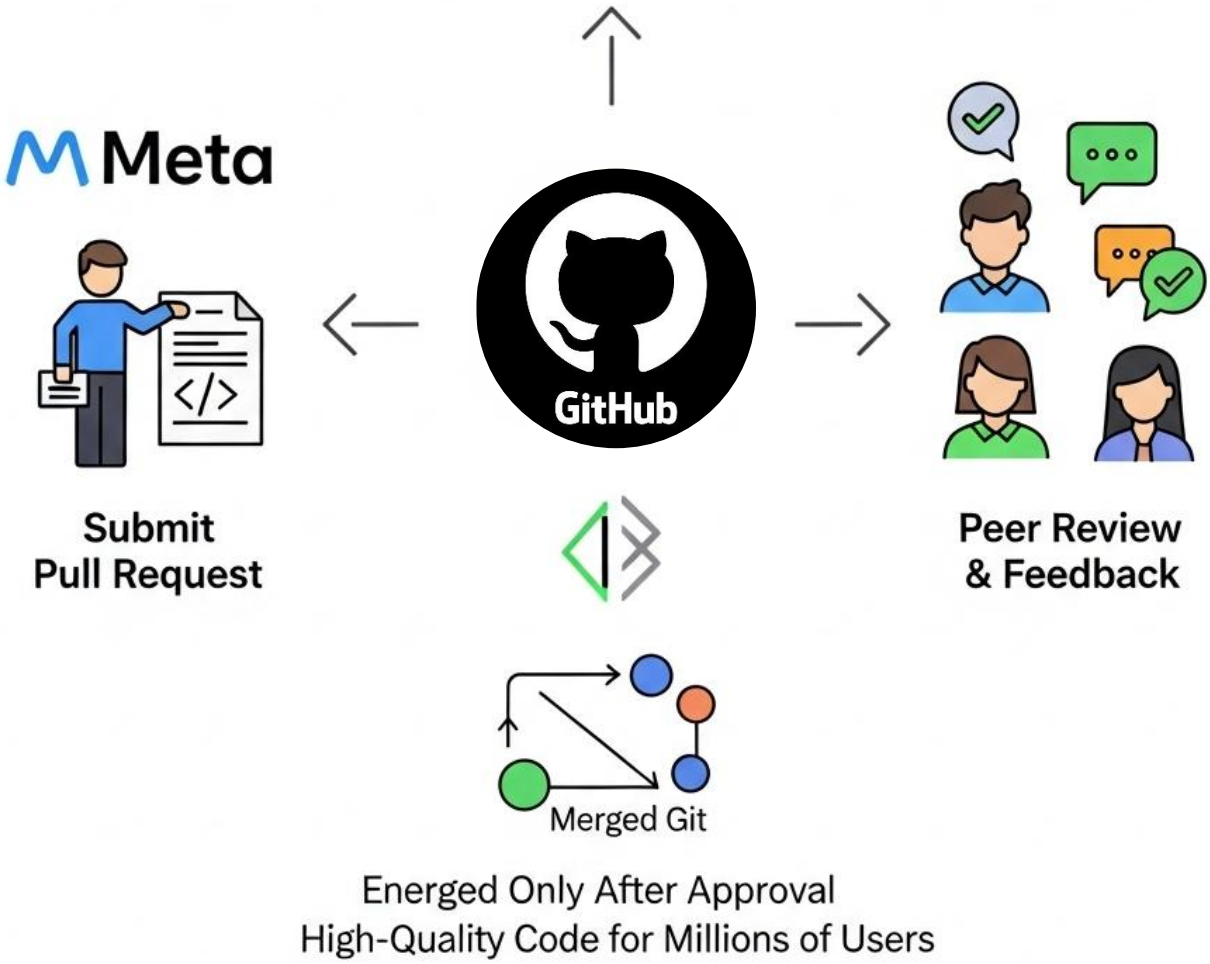
🌍 **Example:**
Google uses Git hooks in large projects to enforce **linting and formatting rules** so code always looks consistent before it's committed.

🆕 12. Code Review & Pull Request Workflows

✅ **Technical Definition:**
The process of submitting changes through pull requests and having peers review code for quality, security, and maintainability.

🎀 **Down-to-Earth Explanation:**
It's like asking a friend to proofread your Instagram caption before posting 📸 — they might catch typos you missed.

🌍 **Example:**
Facebook (Meta) mandates peer reviews on GitHub/Phabricator before merging, ensuring code meets high-quality standards for millions of users.

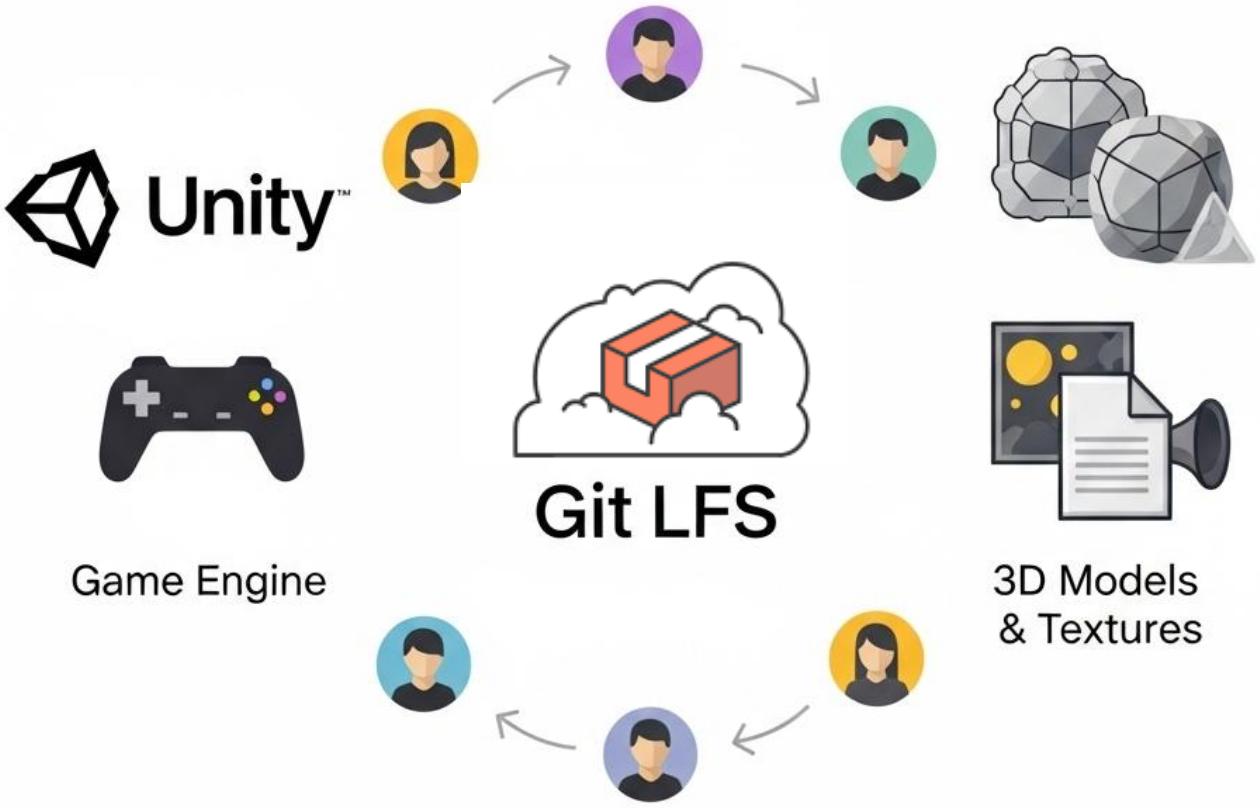


🆕 13. Git LFS (Large File Storage)

✅ **Technical Definition:**
An extension for Git that handles large files (images, videos, datasets) by storing them outside the repo while keeping lightweight pointers inside Git.

🎀 **Down-to-Earth Explanation:**
Imagine moving house 🏠 — instead of stuffing your car with furniture, you just keep the receipts and store the heavy stuff in a warehouse.

🌍 **Example:**
Unity (gaming engine) uses Git LFS to manage large asset files like 3D models and textures shared among developers.



🆕 14. Monorepos & Multi-repos

✅ **Technical Definition:**

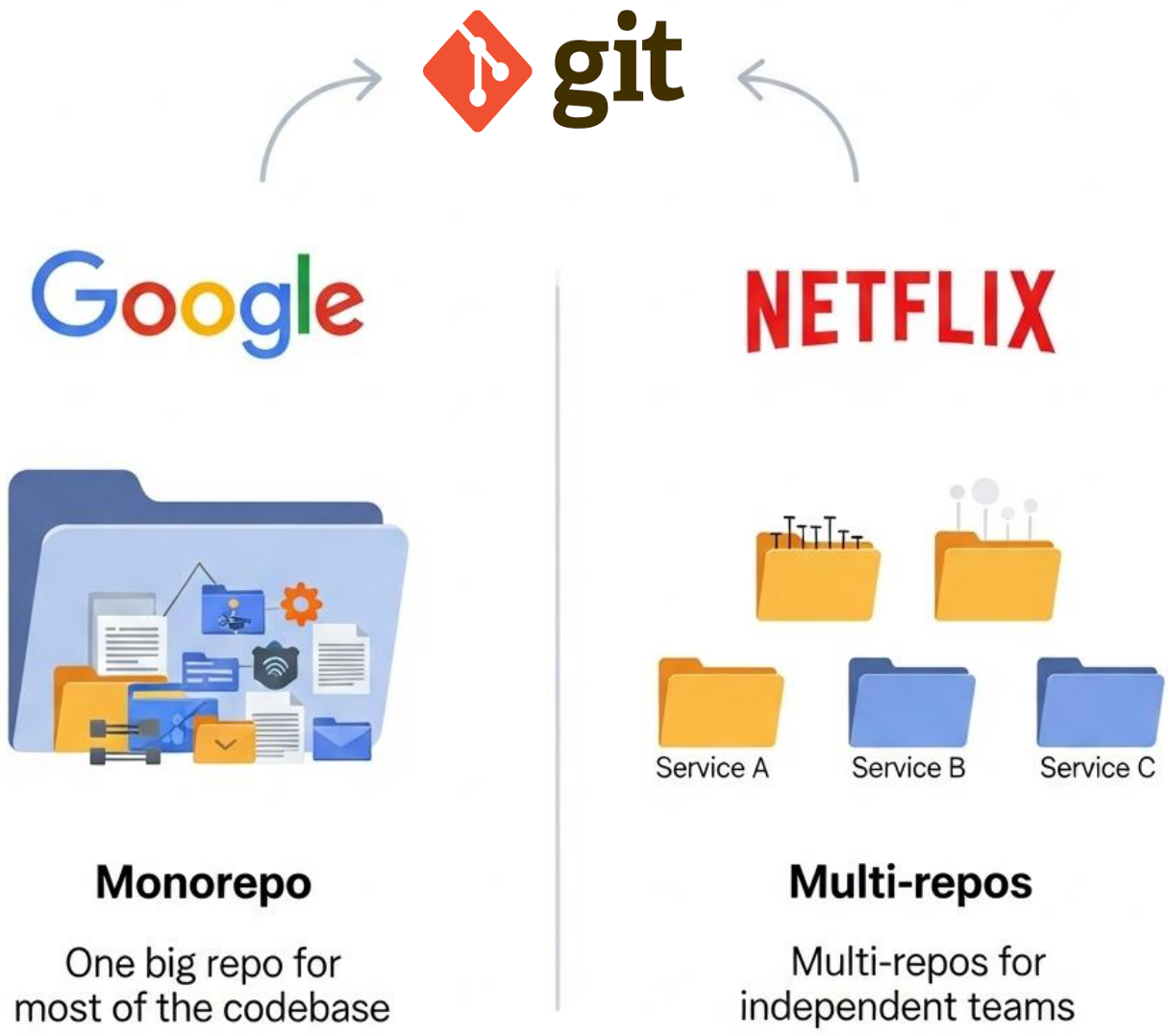
- **Monorepo**: Storing many projects in a single Git repository.
- **Multi-repo**: Each project has its own repository.

🎀 **Down-to-Earth Explanation:**
It's like organizing your closet:

- Monorepo = one big closet with all outfits together 👗
- Multi-repo = separate closets for shoes, dresses, and coats 👠👚🧥

🌍 **Example:**

- **Google** famously uses a **monorepo** for most of its codebase.
- **Netflix** prefers **multi-repos** for different services to keep teams independent.

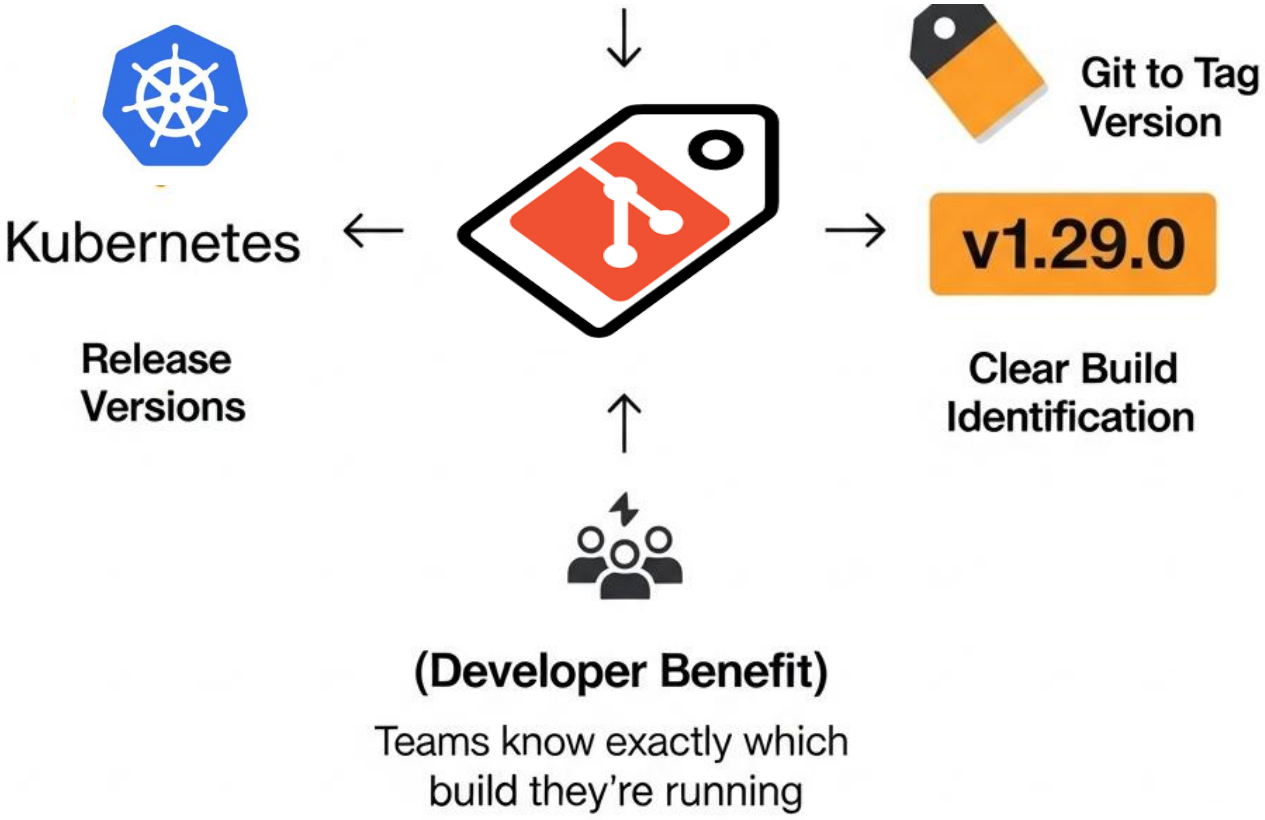


🆕 15. Release Tagging & Versioning

✅ **Technical Definition:**
Tags in Git mark specific commits as important (e.g., release versions). Semantic versioning (v1.2.3) indicates updates, features, and fixes.

🎀 **Down-to-Earth Explanation:**
Like labeling your gym selfies 📸 — "Before," "During," "After." You always know which stage you're looking at.

🌎 **Example:**
Kubernetes uses Git tags for each release version (like `v1.29.0`), so teams know exactly which build they're running.

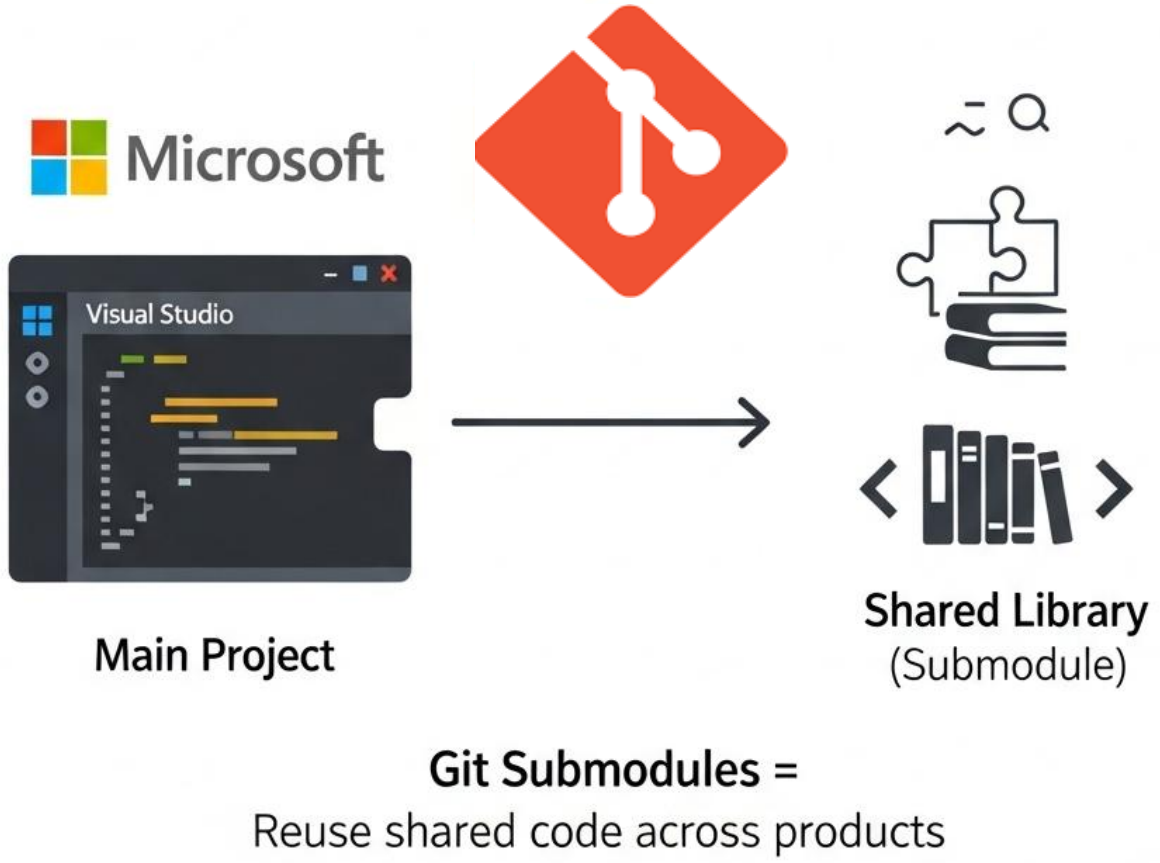


🆕 16. Git Submodules & Subtrees

✅ **Technical Definition:**
Mechanisms to manage one Git repo inside another — useful for projects that depend on shared components.

🎀 **Down-to-Earth Explanation:**
It's like renting an Airbnb with a smaller guest house 🏠 — you manage your main house, but also keep the guest house synced when needed.

🌍 **Example:**
Microsoft uses submodules in some projects to embed shared libraries across products.

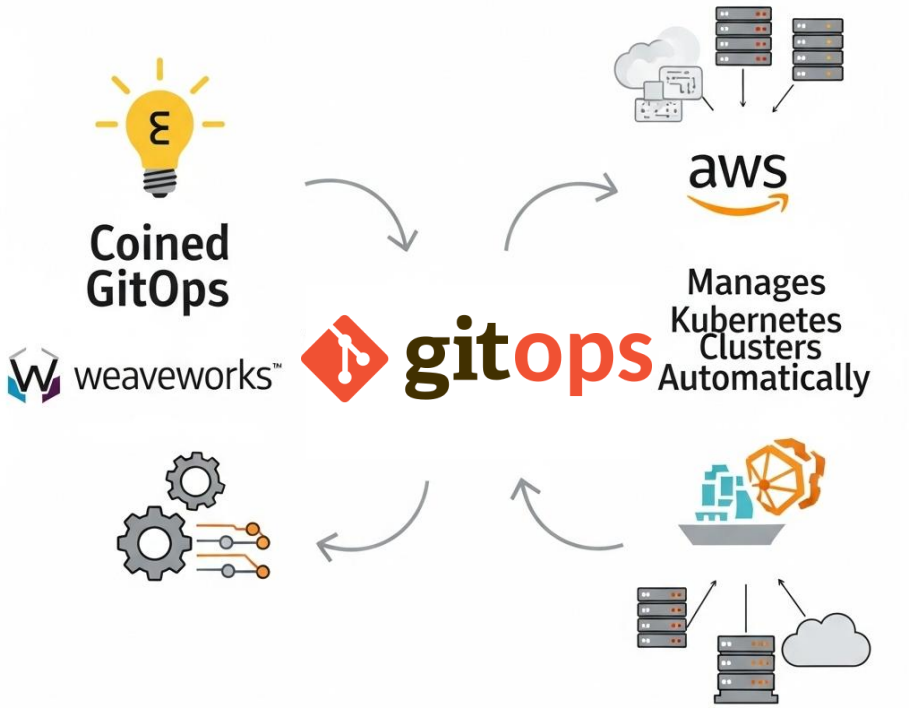


🆕 17. GitOps

✅ **Technical Definition:**
A modern practice where Git repositories act as the single source of truth for infrastructure and application deployments.

🎀 **Down-to-Earth Explanation:**
Like keeping all your diet and workout plans in one shared Google Doc 📝 — every change you make auto-updates your smartwatch and fridge.

🌍 **Example:**
Weaveworks coined GitOps, and **Amazon** adopted it for managing Kubernetes clusters automatically.



🆕 18. Security in Version Control

✅ **Technical Definition:**
Practices like secret scanning, signed commits, and access control to ensure code integrity and prevent leaks.

🎀 **Down-to-Earth Explanation:**
It's like putting locks on your diary 📔 — making sure only trusted friends can write in it, and nobody slips in fake entries.

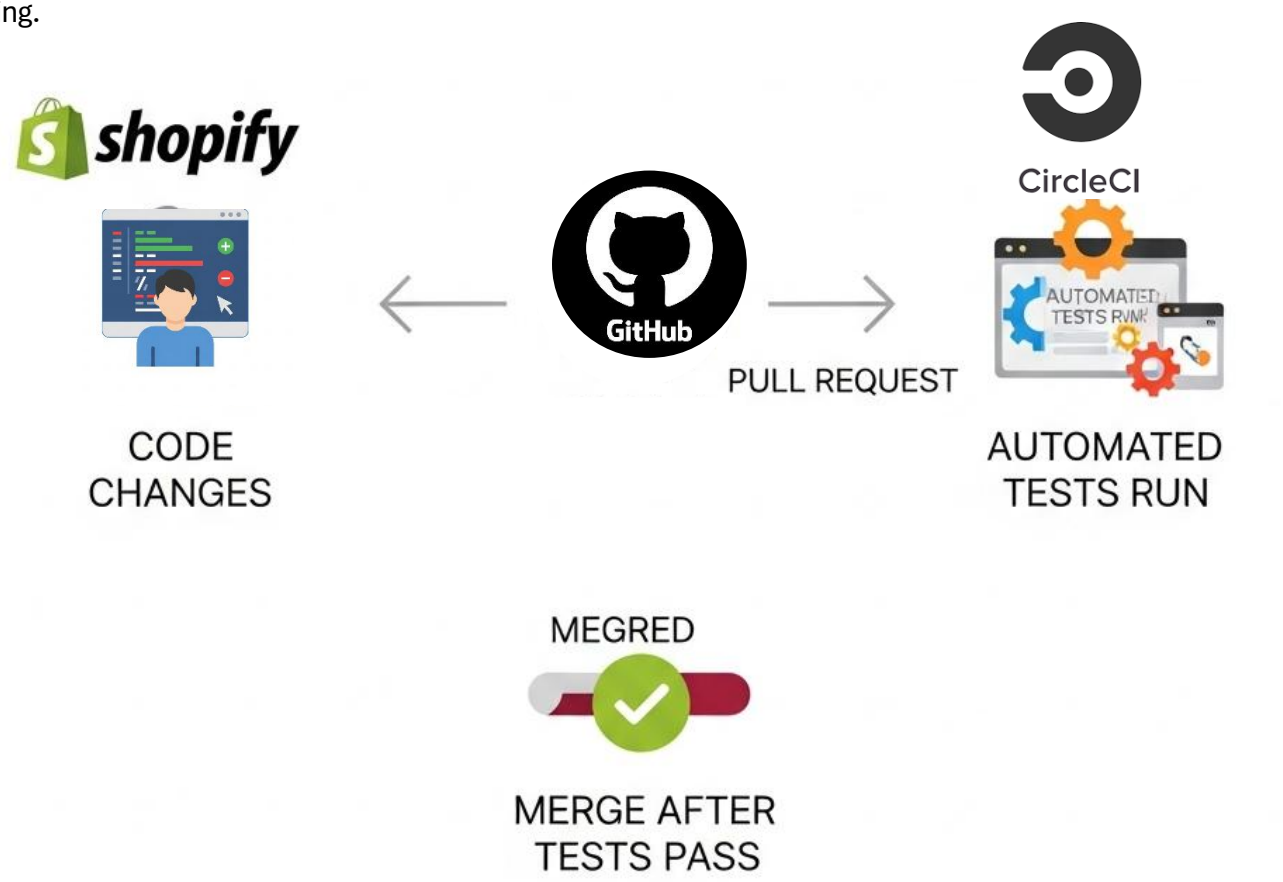
🌍 **Example:**
GitHub runs **secret scanning** across all public repos to prevent accidental leaks of API keys (e.g., AWS credentials).

🆕 19. CI/CD + Version Control Integration

✅ **Technical Definition:**
Connecting Git with pipelines that auto-build, test, and deploy when new code is merged.

🎀 **Down-to-Earth Explanation:**
It's like having a personal trainer 🏋️ who checks your form, logs your workout, and posts your results online instantly every time you finish.

🌍 **Example:**
Shopify integrates GitHub with **CircleCI** pipelines so every pull request runs full automated tests before merging.

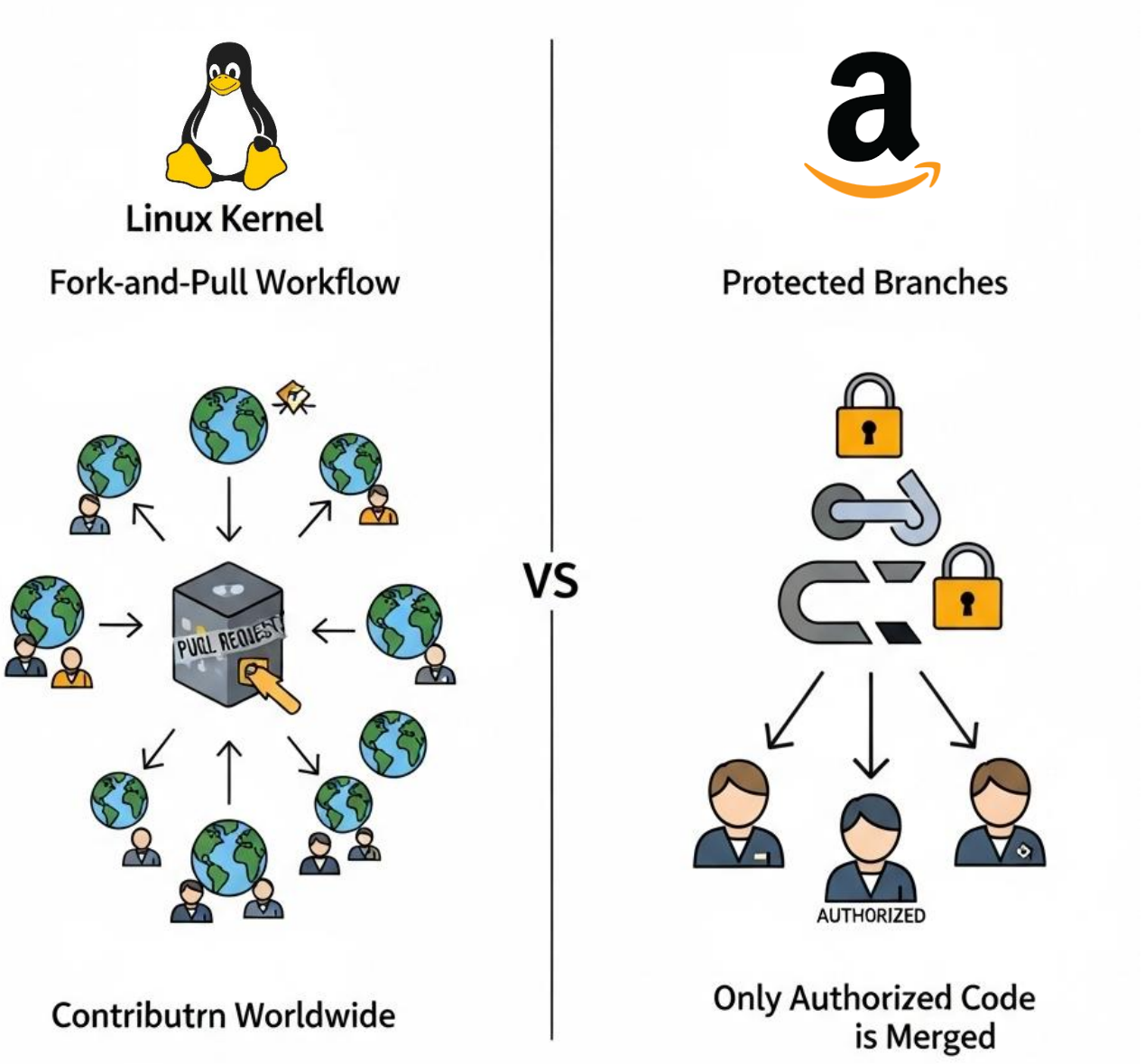


🆕 20. Contribution Models (Open Source vs Enterprise)

✅ **Technical Definition:**
Open-source projects use fork-and-pull workflows, while enterprises often use protected branches and access control.

🎀 **Down-to-Earth Explanation:**
Open source = community potluck 🍲 (anyone can bring a dish).
Enterprise = private dinner party 🍽️ (only invited guests can serve food).

🌍 **Example:**
Linux Kernel development uses a fork-and-pull workflow from contributors worldwide.
Amazon uses protected branches internally for AWS projects to ensure only authorized code is merged.



📌 Key Takeaways

- **Version control is the backbone of modern software development** — it tracks changes, enables teamwork, and prevents overwriting.
- **Core Git concepts** like repos, commits, branches, merges, and rebases make collaboration structured and history clean.
- **Branching strategies** (Feature Branching, Git Flow, Trunk-Based) allow teams to choose workflows that match their speed and stability needs.
- **Collaboration tools** such as Pull Requests and Code Reviews ensure quality and shared ownership of code.
- **Advanced practices** like GitOps, CI/CD integration, monorepos, and Git LFS scale Git for enterprises and cloud-native environments.
- **Real-world adoption** (Microsoft, Google, Netflix, Amazon, Linux Kernel) shows how version control powers projects from open-source to enterprise level.

👉 In short: **Version control** isn't just about **code storage** — it's about c**ollaboration, reliability, and scaling ideas** into products used by millions.

Made by

Chetan Raj

For more such docs, click on the link or scan the barcode
https://archive.org/details/@chetan_raj663

Connect with me on LinkedIn
https://www.linkedin.com/in/chetan-raj-58703b27a/